GPU accelerated lattice Boltzmann modeling for simulation of bubble dynamics

Rou Chen[a], Huidan(Whitney) Yu[a,*]

[a]*Department of Mechanical & Energy Engineering, Indiana University-Purdue University Indianapolis, Indianapolis (IUPUI), IN 46202, USA*

Abstract

Massive parametrization and classification are required to investigate the underlying physics in bubble dynamics numerically, which leads the computational cost becoming high. The suitability of the lattice Boltzmann method for GPU (Graphics Processing Unit) parallel computation provides an opportunity to overcome it. In this paper, we perform GPU parallelization and optimization for the free energy-based lattice Boltzmann method in the multi-phase flow. Coalescence-induced bubble detachment is used as an application case which involves liquid-gas and liquid-gas-solid interactions. Physical accuracy and computational efficiency of the model are two primary sides to be tested. The reliability of the model is validated by a bubble of half-sphere sitting on the different wetting solid surfaces with the targeted contact angle from 40° to 150°. The performance of the application case is tested by using approaches with GPU cards. It shows the performance by NVIDIA Tesla P100 is respectively up to 20, 2110 faster than NVIDIA Tesla C2050/C2075 and serial counterpart (Intel Broadwell E5-2683 v4). For the multi-GPU implementation, the acceleration is increased by the numbers of the GPU card when the resolution is high.

Keywords: lattice Boltzmann method; coalescence-induced bubble detachment; GPU parallel computing; multi-GPU

1. Introduction

The GPU acceleration for LBM has a short history, but it develops rapidly due to the increasing demands from the computation of complex flows, especially when complex geometry, multi-scale, multi-phased, and massive parametric study is involved. The first work applying GPU acceleration for LBM[**?**] was in 2003. Li *et al.* performed computations, in which textures and render buffers were used. Based on this approach, Zhu *et al.* developed GPU parallelism for two miscible fluids systems and achieved speed three times faster than serial CPU executions. In 2006, a parallel computing platform and application programming interface

*Corresponding author
Email address: `whyu@iupui.edu` (Huidan(Whitney) Yu)

(API) model created by NVIDIA[?] called CUDA was launched. The first parallel implementation of D3Q19 on CUDA was done in 2008 [?], a sequential code (470.lbm) parallelized from the SPEC CPU2006[?] on CUDA. After that, the GPU acceleration for LBM in the stage of general-purpose GPU (GPGPU) implementations and optimization with CUDA. Optimization of GPU acceleration for LBM has been developed in three aspects: memory access patterns, register utilization, and overlap of memory transfer and arithmetic operations.

Memory access pattern attracts the most attention in the literature through propagation schemes, data layout, and memory addressing. *Propagation schemes:* GPU parallelization of LBM mainly consists of two parts in each iteration: collision and streaming. Since the collision occurs at the local nodes but steaming requires information from neighboring nodes, the efforts on the fast access to the memory have been on the development of propagation schemes[?] for the streaming part. Shift algorithm with shared memory [? ?] was applied at the beginning to avoid misaligned accessing at the expense of adding an extra kernel to exchange data through GeForce 8800 Ultra. The propagation scheme of the A-A pattern (only one set of distribution functions (DFs) in memory) based on the shift algorithm with shared memory was developed[?] in 2009. The A-A pattern reduced GPU memory by 50% compared with the A–B pattern (two sets of DFs in memory). Shortly after, split propagation scheme (misaligned write) and reversed propagation scheme (misaligned read) [?] were developed, which have a 15% improvement compared with a shared memory scheme with the GTX 295. *Data layout*: Two popular schemes of data layout include array of structure (AoS) and structure of array (SoA). SoA switches from AoS for the optimization of GPU parallelism[?]. It has 7-10 times speed increasing with Tesla Kepler K20[?]. Herschlag *et al*[?] applied more complex data layouts, i.e., collected SoA in the GPU architectures and accelerated computational speed by 5-20% compared with SoA layout. *Memory addressing*: Indirect addressing and semi-addressing are two common patterns of memory addressing that is the original approach for CPUs. Compared with direct address[?], they have 4-5 times acceleration for complex geometry problems.

Two other aspects are much less addressed. Accessing register memory consumes zero clock cycles per instruction comparing to 400-600 cycles for global memory. Whereas, the amount of register is limited. An NVIDIA compiler provides a flag to limit the register usages for register utilization[?]. Meanwhile, there are two strategies for overlap of memory transfer and arithmetic operations, including tiling the 3D lattice grid into smaller 3D blocks and branch divergence removal[?].

In the case of multi-phase flows, similar approaches of optimization are applied in different models. As reviewed [?]. the performance of LEE model[?], Shan-Chen(SC) model[?], and Free Energy(FE) model[?] are 125.1, 210, 238 (MLUPS) respectively with C2050 GPU cards in the float precision, where million lattice update per seconds (MLUPS) indicates the performance of lattice Boltzmann simulations. Meanwhile, GPU hardware has been developed quickly. For example, MLUPSs have increased 1.3 times with Tesla Kepler K20 and 1.7 times with Tesla Kepler K40, respectively comparing with C2050[?]. In the LEE model, Tesla P100 accelerated 10 and 680 times compared with Tesla M2070, serial CPU(i7-4930K), respectively[?].

Development of the parallelization combined with GPU, CPU, and network is popular in recent years to solve multi-scale, multi-physics research problems. The parallelization include multi-GPU[? ?], LBM code with CUDA and OpenMP for multi-GPU clusters[?], LBM with CUDA and MPI [?], and a multi-node MPI/OpenMP LBM code with OpenACC[?].

In this paper, we perform GPU parallelization and optimization for free energy-based lattice Boltzmann method in the multi-phase flow with liquid-gas and liquid-gas-solid interactions. This multi-phase LBM model is developed and refined by Lee's group [? ? ? ?] which has verified as a reliable model for high-density ratio simulations. Meanwhile, combining features of GPU with LBM is significant in achieving an efficient GPU-LBM algorithm. An A-B (2 sets of distribution functions) pattern, dynamic allocation, data layout, propagation scheme, register utilization, branch divergence removal, and multi-GPU are employed in this paper. A half sphere bubble sitting on the nine different wetting solid surfaces with targeted contact angels varying from 40° to 150° is applied to validate the reliability of the model. After the validation, the performance of the model is tested by the application case that a bubble detaches from the solid surface caused by two bubbles coalesce. Serial CPU, single GPU, multi-GPU are three different ways to test the efficiency of the model. Finally, a summary discussion is provided.

2. Computational methodology

2.1. Lattice Boltzmann modeling for multi-phase flows

2.1.1. Governing equations

The governing equations for multi-phase flow including continuity, pressure evolution and momentum equation. A diffuse interface is applied to separate gas and liquid as a modeling of multi-phase flows thus the continuity equation can be written as Cahn-Hilliard equation, resulting in the following governing equations:

$$\partial C/\partial t + \mathbf{u} \cdot \nabla C = \nabla \cdot (M \nabla \mu) \tag{1}$$

$$\partial p/\partial t + \rho c_s^2 \nabla \cdot \mathbf{u} = 0 \tag{2}$$

$$\rho(\partial \mathbf{u}/\partial t + \mathbf{u} \cdot \nabla \mathbf{u}) = -\nabla p + \mu \nabla C + \nabla \cdot \eta(\nabla \mathbf{u} + (\nabla \mathbf{u})^T) \tag{3}$$

with $M(>0)$ the mobility [?], C the composition, p the hydrodynamic pressure, and μ the chemical potential. The chemical potential is formulated as $\mu = \mu_0 - \kappa \nabla^2 C$ in which μ_0 is the classical part of the chemical potential. In the vicinity of the critical point, simplification of van der Waals equation of state can be made[?] for the control of interface thickness and surface tension at equilibrium. In this case, we assume that the bulk energy E_0 takes a form[?] of $E_0 = \beta C^2(C-1)^2$ with β an amplitude to control the interaction energy. As a result, $\mu_0 = \partial E_0/\partial C = 2\beta C(C-1)(2C-1)$. In an interface at equilibrium, the interface profile is $C(z) = 0.5 + 0.5\tanh(2z/D)$ where z is the distance normal to the interface and D is the numerical interface thickness, which is chosen based on accuracy and stability. Given D and β, one can compute the gradient parameter $\kappa = \beta D^2/8$ and the surface tension force $\sigma = \sqrt{2\kappa\beta}/6$.

2.1.2. Lattice Boltzmann equations

The lattice Boltzmann equation push (LBE) (before the time discretization) for the binary flow is given as following [?]

$$\partial f_\alpha/\partial t + \mathbf{e}_\alpha \cdot \nabla f_\alpha = -(f_\alpha - f_\alpha^{eq})/\lambda + \frac{3}{c^2}(\mathbf{e}_\alpha - \mathbf{u}) \cdot \mathbf{F} f_\alpha^{eq} \tag{4}$$

where $\mathbf{F}(= \frac{1}{3}\nabla\rho c^2 - \nabla p - C\nabla\mu)$ is the intermolecular force between liquid and gas on the interface, f_α is the particle distribution function with discrete molecular velocity $\mathbf{e}_\alpha$ along the α-th direction and λ is the relaxation time related to the kinematic viscosity $\nu = \frac{1}{3}c^2\lambda$. The equilibrium distribution function is defined as $f_\alpha^{eq} = \rho\omega_\alpha[1+3(\mathbf{e}_\alpha\cdot\mathbf{u})/c^2+9(\mathbf{e}_\alpha\cdot\mathbf{u})^2/(2c^4)-3\mathbf{u}^2/(2c^2)]$ where ω_α is the weight associated with a particular discretized velocity $\mathbf{e}_\alpha$, ρ and $\mathbf{u}$ are macroscopic density and velocity respectively, and $c = \delta x/\delta t = 1$ in lattice units (i.e., $\delta t = \delta x = 1$).

Defining a new particle distribution function $g_\alpha = \frac{1}{3}f_\alpha c^2 + (p - \frac{1}{3}\rho c^2)\Gamma_\alpha(0)$ in which $\Gamma_\alpha(\mathbf{u}) = f_\alpha^{eq}/\rho$ and taking the total derivative $D_t = \partial_t + \mathbf{e}_\alpha \cdot \nabla$ of g_α and along characteristics over the time step δt result in

$$\bar{g}_\alpha(\mathbf{x}+\mathbf{e}_\alpha\delta t, t+\delta t) = \bar{g}_\alpha(\mathbf{x},t) - \frac{\bar{g}_\alpha - \bar{g}_\alpha^{eq}}{\tau+0.5}|_{(\mathbf{x},t)} + (\mathbf{e}_\alpha - \mathbf{u})\cdot[\frac{1}{3}\delta t\nabla^{MD}\rho c^2(\Gamma_\alpha(\mathbf{u}) - \Gamma_\alpha(0)) - C\delta t\nabla^{MD}\mu\Gamma_\alpha]|_{(\mathbf{x},t)} \tag{5}$$

where ∇^{MD} and ∇^{CD} are referred to mixed difference approximation and central difference approximation respectively [?] and $\tau(= \lambda/\delta t)$ is the non-dimensional relaxation time.

The momentum and hydrodynamic pressure are the zeroth and first-order moment of $\bar{g}_\alpha$, computed as $\rho u = \frac{3}{c^2}\sum \mathbf{e}_\alpha\bar{g}_\alpha - \frac{\delta t}{2}C\nabla^{CD}\mu$ and $p = \sum \bar{g}_\alpha + \frac{\delta t}{6}\mathbf{u}\cdot\nabla^{CD}\rho c^2$

For the transformation of the composition C, a second distribution function is introduced in a simple format of $h_\alpha = (C/\rho)f_\alpha$ and $h_\alpha^{eq} = (C/\rho)f_\alpha^{eq}$. Similarly, taking the total derivative D_t of h_α yield

$$\bar{h}_\alpha(\mathbf{x}+\mathbf{e}_\alpha\delta t, t+\delta t) = \bar{h}(\mathbf{x},t) - \frac{\bar{h}_\alpha - \bar{h}_\alpha^{eq}|_{(\mathbf{x},t)}}{\tau+0.5} + \delta t(\mathbf{e}_\alpha - \mathbf{u})\cdot[\nabla^{MD}C - \frac{3C}{\rho c^2}(\nabla^{MD}p + C\nabla^{MD}\mu)]\Gamma_\alpha|_{(\mathbf{x},t)} + \delta t M\nabla^2\mu\Gamma_\alpha|_{(\mathbf{x},t)} \tag{6}$$

The composition C is the zeroth-order moment of $\bar{h}_\alpha$ computed as $C = \sum_\alpha \bar{h}_\alpha + 0.5\delta t M\nabla^2\mu$. The density ρ and the dimensionless relaxation frequency $(1/\tau)$ are taken as linear functions of the composition by $\rho(C) = C\rho_1 + (1-C)\rho_2$ and $1/\tau(C) = C/\tau_1 + (1-C)/\tau_2$.

2.1.3. Boundary conditions

Considering the liquid-gas-solid interaction for lattice Boltzmann modeling of multi-phase flow, four different types of boundary conditions are showing as following:

1. Unknown particle distribution functions at boundary nodes
   Bounce-back scheme i.e. $g_\alpha(\mathbf{x_s}, t) = g_\alpha(\mathbf{x_f}, t)$ and $h_\alpha(\mathbf{x_s}, t) = h_\alpha(\mathbf{x_f}, t)$, $\mathbf{x_s}$ is the node on the boundary, $\mathbf{x_f} = \mathbf{x_s} - \mathbf{e}_\alpha \delta t$ is the node in the fluid.
2. The boundary for the $\nabla\phi$
   To prevent unphysical mass and momentum transfer through the boundary nodes. ϕ is macroscopic variables such as p, C, μ. The no flux condition is presented as $e_\alpha \dot{\nabla}\phi|_s = 0$. Discrete the no flux condition, it can be simplified as $\phi(\mathbf{x_s} + \mathbf{e}_\alpha \delta t) = \phi(\mathbf{x_s} - \mathbf{e}_\alpha \delta t)$, $\phi(\mathbf{x_s} + 2\mathbf{e}_\alpha \delta t) = \phi(\mathbf{x_s} - 2\mathbf{e}_\alpha \delta t)$, where the points $(\mathbf{x_s} + \mathbf{e}_\alpha \delta)$, $(\mathbf{x_s} + 2\mathbf{e}_\alpha \delta)$ are in the fluid domain, the points $(\mathbf{x_s} - \mathbf{e}_\alpha \delta)$, $(\mathbf{x_s} - 2\mathbf{e}_\alpha \delta)$ are out of the fluid domain.
3. The boundary for $\nabla^2\mu$
   To ensures no mass flux in the normal direction of the solid wall, i.e
   $$\mathbf{n} \cdot \nabla\mu|_s = 0. \tag{7}$$
4. The boundary for $\nabla^2 C$
   It can be derived from minimizing the free surface energy $\Psi_s = \int_S (\phi_0 - \phi_1 C_s + \phi_2 C_s^2 - \phi_3 C_s^3 + \ldots) dS$ caused by the interactions between the liquid-gas interface and solid surface.
   Minimizing the total energy by calculus of variations[**?**] leads to
   $$\mathbf{n} \cdot \nabla C|_s = d\Psi_s / dC_s \tag{8}$$
   $\mathbf{n}$ is the unit vector in the normal direction of the solid wall. Retain higher-order terms in Ψ_s can avoid the numerical instability. Hence, the cubic boundary condition is selected[**?**] in this work, the condition for solid wall i.e. Eq.8 can be simplified as
   $$\mathbf{n} \cdot \nabla C|_s = -\phi_c/\kappa(C_s - C_s^2), \tag{9}$$
   $\phi_c = \Omega_c\sqrt{2\beta\kappa}$. Here Ω_c is the wetting potential which related to the contact angle α i.e. $\Omega_c = cos\alpha$.

2.2. *GPU parallelism and optimization*

The GPU acceleration has been practiced in our research group for a couple years for different research areas such as turbulence[?], biomedical flows[?] and porous media flows [? ?]. The general flow chart of GPU implementation is indicated in Fig. 1. The difference between GPU parallel and CPU serial is the device used for the LBM implementation. It is important to combine the feature of GPU with LBM to achieve an efficient GPU-LBM algorithm. Firstly, an A-B (2 sets of distribution functions) pattern is applied before and after streaming to avoid data dependency. Then developed CUDA parallel algorithms are employed in our simulation that includes dynamic allocation, data layout, propagation scheme, register utilization, branch divergence removal, and multi-GPU.

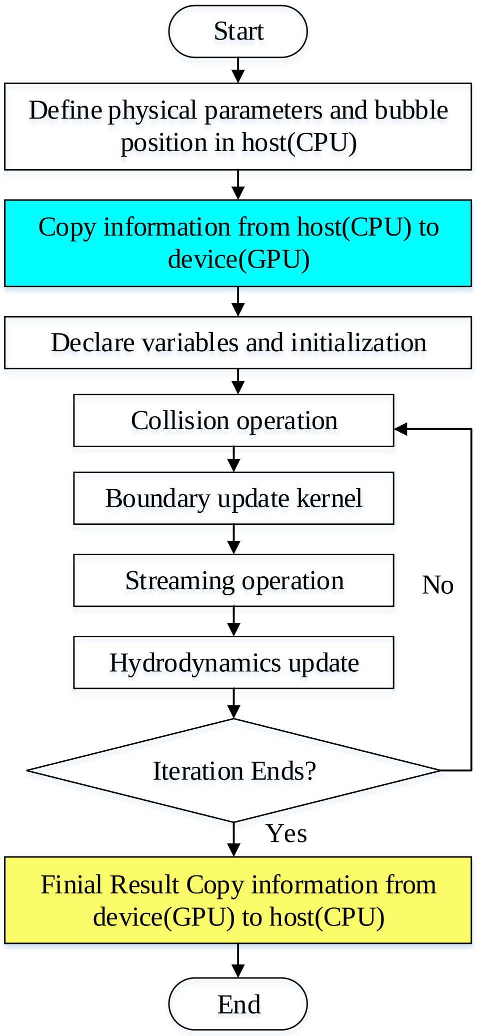

Start
Define physical parameters and bubble position in host(CPU)
Copy information from host(CPU) to device(GPU)
Declare variables and initialization
Collision operation
Boundary update kernel
Streaming operation
Hydrodynamics update
Iteration Ends?
No
Yes
Finial Result Copy information from device(GPU) to host(CPU)
End

Figure 1: Flow chart of GPU Implementation

2.2.1. *Dynamic Allocation*

By default, the compiled model for a code has a characteristic that the instructions and parameters of the code must be linked in the 2GB static continuous address space. This limitation takes the dominant effect in impeding the simulation of large scale problems. For example, two sets of distribution functions of the lattice Boltzmann model and flow properties in the multi-phase flow, including density, velocity, pressure, chemical potential, and composition are essential arrays to be defined in the CPU host. From the experience of investigating bubble dynamics in a three-dimension (3D) model, the mesh size is around 220×120×220 for our calculations. Due to the limitation of 2GB static continuous address space, the mesh size is up to 280×140×280, which is close to the appropriate mesh size. All these experiences are based on the two bubble coalescence. However, a multiply bubble system, especially with the complex geometry, the needed memory must be larger than 2GB. Nowadays, a single NVIDIA GPU card has up to 12GB of memory for computing. Hence, it is critical to find a way to release this limitation. Dynamic memory (heap) instead of a static memory (stack) was applied in our simulations. Dynamic memory (heap) randomly allocates portions from a large pool of memory. The transmission array should be logically contiguous when the data copy between CPU (host) to GPU (device). Hence, in our algorithm, a class of the variable has been developed.

- The class can dynamically allocate memory in one dimension array, and each variable only uses a total of 5×sizeof(float) or sizeof(double) on the stack. Each variable includes three dimensions in different directions, one variable of fluid properties, and data precision. The illustration likes Array3<precision; 3 dimensions in different directions> a variable of fluid property. Take composition C as an example: the class is Array3<double; nx; ny; nz> C.
- All the other memory is allocated dynamically on the heap in orders in a continuous block through properly overloading the copy constructor. The information which is related to assignment operators will be swap and destroyed.

This class is to manage allocation, deallocation of memory, and utilize the full memory capacity of the GPU cards easily.

2.2.2. Propagation Scheme

Propagation in the LBM relates to streaming. During streaming, there is a problem called misaligned access. There are two main strategies to address this issue in literature as 'push' and 'pull' algorithms depending on the streaming direction, which is shown in Fig. 2 and 3 respectively. The streaming direction of the 'push' scheme is the same as the usual streaming, which pushes particles from the current cell to the adjacent cells. In other words, it has aligned read and misaligned write in the implementation. The distribution function after the collision is stored in a lattice, as Fig. 2(a) indicates, then the post-streaming values push to the adjacent cell as Fig. 2(b) shows. Whereas the 'pull' scheme has opposite streaming direction, it pulls particles from the adjacent cells to the current cell. In the implementation, it yields to misaligned read and aligned write. The distribution function read from the adjacent cells, as Fig. 3(a) shows. After streaming, the values propagate to the local cell, as Fig. 3(b) illustrates. Since misaligned read is faster than misaligned write in the GPU[**?**], the 'pull' scheme is employed in our simulation.

2.2.3. Data Layout

Coalesced memory access is an efficient way to reduce the memory latency of a GPU program that results in acceleration. Coalesced memory access means all the threads in a block access memory address at the same time. Hence, the multiple dimensions arrays are expanded into one dimension arrays. We take the D3Q27 model and assume the mesh size $nx \times ny \times nz$ to show the data layout in the GPU. Regularly, the distribution function stores as 4D array on the CPU such as $f[x][y][z][i]$. Here x, y, z represent the space; i ranges from 0 to 26. Different structures of one-dimensional arrays in the GPU leads to different effects in

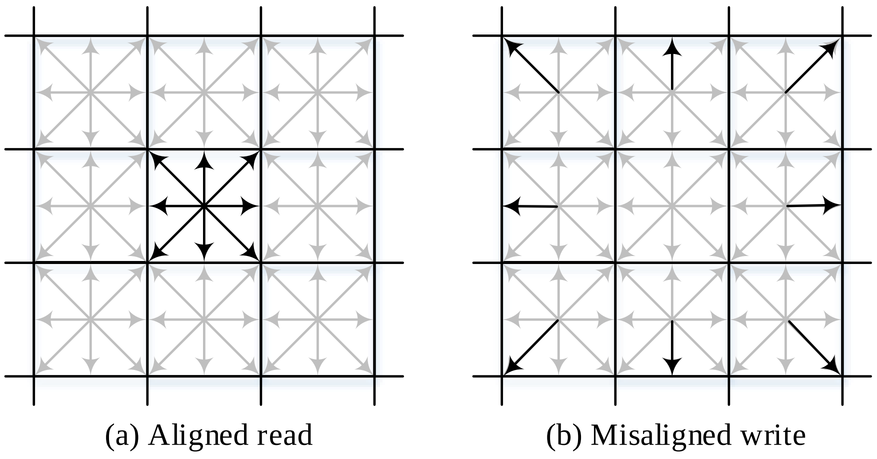


Figure 2: Push scheme for propagation (a) the state before streaming with aligned read (b) the state after streaming with misaligned write

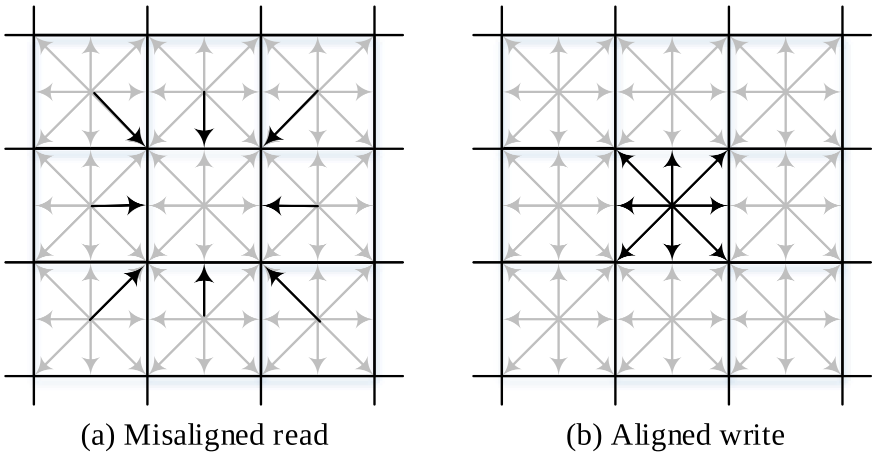


Figure 3: Pull scheme for propagation (a) the state before streaming with misaligned read (b) the state after streaming with aligned write

the GPU acceleration. There are two main schemes for data layout: Array of Structure (AoS) and Structure of Array (SoA).

- Array of Structure (AoS)

  Twenty-seven distributions of each cell are arranged 27 consecutive elements of the 1D array illustrated in the top panel of Fig. 4. The one dimension array format is $f[z \times ny \times nx \times 27 + y \times nx \times 27 + z \times 27 + i]$. This schema is preferred in the CPU parallelization.

- Structure of Array (SoA)

  The value of one distribution of all cells in whole computational domain occupies consecutive elements in memory, which is indicated in the bottom panel of Fig. 4. The distribution functions are addressed as $f[i \times z \times nz \times ny \times nx + z \times ny \times nx + y \times nx + z]$. In this scheme, the number of the thread of distribution functions within a wrap (32 threads) can access consecutive memory, so it has been reported that the SoA scheme is suitable for the GPU acceleration[?].

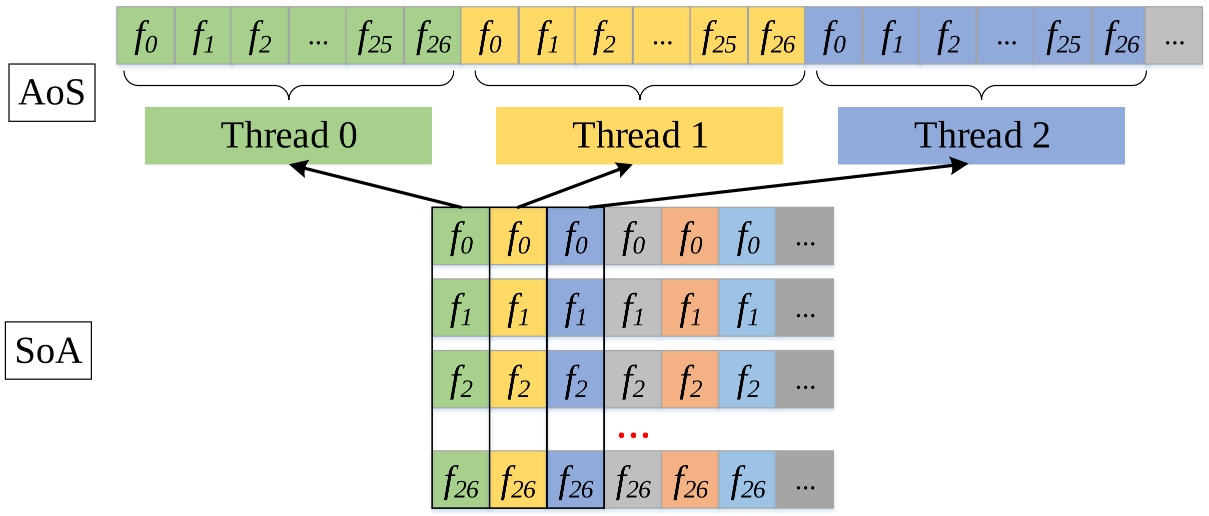


Figure 4: Data layout: AoS and SoA

2.2.4. *Register Utilization*

CUDA provides a memory hierarchy, including device memory and on-chip memory. Our attention is mainly on the global memory, register, shared memory, and local memory. Global memory is the main part of the device memory, which has a large storage memory and a high latency of 400–600 cycles[?]. The register is a kind of on-chip resource distributed to each thread, which has nearly zero cycle latency. But the number of registers may limit the number of concurrent threads on each SM. Similar to the register; shared memory can be fast accessed but limited by its upper bound of storage. When registers or shared memory spills, the extra data will be stored in local memory. The latency of this type of memory is similar to global memory.

In these kinds of conditions, register memory is better to be applied in the optimization due to the low latency. Whereas the amount of registers is limited, a single streaming multiprocessor contains 65,536 registers for Tesla K20, and the memory for each register is 32-bit. While too many registers are assigned in each thread, it can't take full utilization in the memory. For example, 44 registers are needed in the D3Q27 per thread and 1024 threads in one block. 44K registers will be used in one block. In this case, the remaining 21K registers are unused, and only one block is applied in the simulation, then the registers get approximate 60% utilization. The utilization of register memory effectively in each thread is critical in GPU optimization. The selection of the grid and block size is a scheme to get higher register memory utilization.

2.2.5. Branch Divergence Removal

During the execution, all 32 threads inside one block sequentially compose a warp. When one warp of threads is waiting for data transfer, other warps, which are ready for executing instructions, would be selected out and executed. Therefore the time spent for computation and data access can be overlapped, leading to latency hiding. For execution, CUDA adopts single instruction with multiple threads of architecture where groups of up to eight threads will execute the same instruction in a thread processing array. If the threads inside a warp are not on the same executive path, some data dependence of the if-else branch will be involved. In this case, the implementation of different branches has to be serialized. The update of boundary condition is realized in another kernel to avoid using if-statements. Due to the stretch direction changes at the boundary, the choice of block size will be changed.

2.2.6. Multi-GPU

The multi-GPU part of my research is in the infant stage. Current programming is based on the basic idea of the multi-GPU. However, the performance improvement of the GPU acceleration is obvious in the problems which need high resolution. The basic idea of the multi-GPU is splitting the total memory into several parts, which depends on the number of GPU cards, and then read and written with these cards. Before doing the multi-GPU parallel, two properties of the device should be checked. Firstly, can the device be able to do peer to peer communication (P2P)? And then, how many GPU cards can be applied in a calculation? As Fig. 5 shows, there are three computational data regions in each GPU card, including internal (white region), halo (orange region), and padding part (green region). First, compute the internal and halo parts in a stream, then exchange halo data between neighboring GPUs in a different stream to update the data in different GPU devices. For example, the halo regions in GPU 0 will be transferred to the beginning padding part of GPU 1, and the padding part in the GPU 0 will get the information in the halo regions in the GPU 1. Next, synchronize computation on all devices before proceeding to the next iteration by the function of *cudaDeviceSynchronize()*. Last, the data from all the devices will be written in the CPU part ignore the

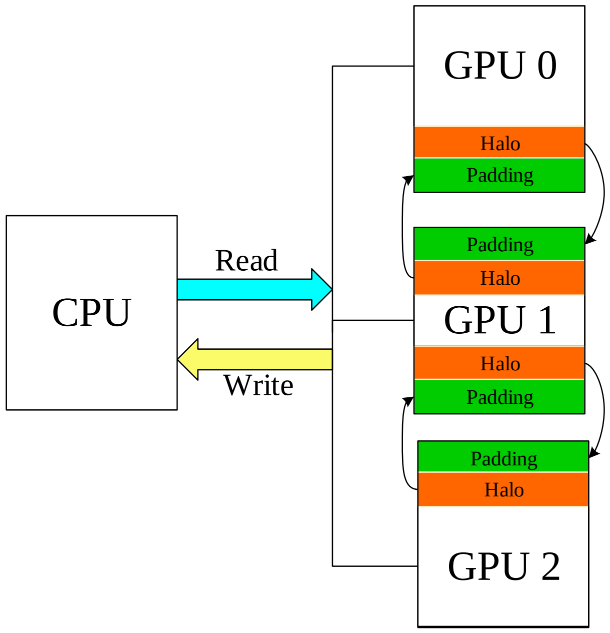


Figure 5: A schematic of data transfer between GPUs

padding part.

3. Application studies

In order to study the bubble dynamics for the coalescence-induced bubble detachment, the computational model is schematized in Fig. 6. Two micro oxygen bubbles (gray color) with the large radius R_F and a small radius R_M in the hydrogen peroxide solution merge into one single bubble (gray color with red color outline) with radius R_e. The density and viscosity of oxygen and hydrogen peroxide are $\rho_{O_2} = 1.3kg/m^3$, $\rho_{H_2O_2} = 1060kg/m^3$, $\eta_{O_2} = 1.92\times10^{-5}kg/(m\cdot s)$, $\eta_{H_2O_2} = 1.06\times10^{-3}kg/(m\cdot s)$ respectively. The numerical study is conducted in a cuboid domain with height $H = 144(\mu m)$, length of the channel L $264\mu m$, and the width of the channel W the same as the length. For top and bottom sides, boundary conditions shown in Sec.2.1.3 is imposed. The periodic boundary is used on the remaining four sides.

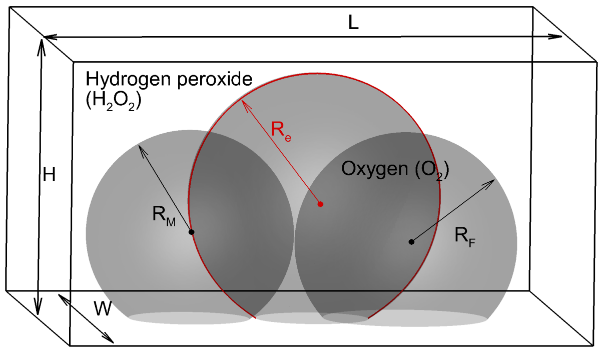


Figure 6: Schematics for coalescence of two micro oxygen bubbles(gray) toward a coalesced bubble (gray with red outline) in the hydrogen solution with a cuboid domain.

3.1. Convergence check and validation

Finding the appropriate spatial resolution for the numerical analysis is the first step. Here, we simulate the contact angel when the bubble with $R_{\alpha^{initial}=90^\circ} = 60\mu m$ initially sitting on a solid surface, and the targeted contact angle of the solid surface is 60°. Fig.7(a) shows the initial bubble (gray color) with half-sphere and the equilibrium state(gray color with the red outline). The simulated contact angle α^{eq} is calculated by Eq.10 through the measurement of the height of liquid-gas interface a and a base diameter of a bubble b on a solid surface at equilibrium state. The liquid-gas interface which is represented by a contour level $C = 0.5$.

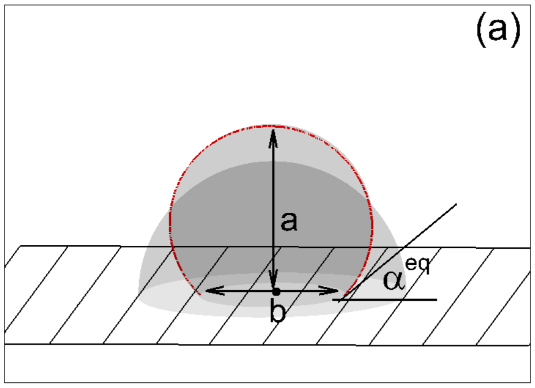
$$\alpha^{eq} = 2arctan(\frac{b}{2a}). \tag{10}$$

The convergence check is listed in the Tab.1. The spatial resolution increases from $44\times24\times44$ to $264\times144\times264$ with six different levels, in which the last row shows the spatial convergence. Considering computational cost and accuracy, we select the spatial resolution of $220 \times 120 \times 220$ to conduct the test.

Resolution	$44 \times 24 \times 44$	$88 \times 48 \times 88$	$132 \times 72 \times 132$	$176 \times 96 \times 176$	$220 \times 120 \times 220$	$264 \times 144 \times 264$
Simulated contact angle	53.00	56.88	58.10	59.22	59.58	59.84
Convergence (%)		6.82	2.10	1.89	0.60	0.43

Table 1: Convergence check when the spatial resolution varies from $44 \times 24 \times 44$ to $264 \times 144 \times 264$ with six levels.

Based on the appropriate resolution, the validation is tested by the case that the initial half-sphere with $60\mu m$ radius on different wetting solid surfaces (different targeted contact angle). Fig.7(b) presents the relationship between the contact angle and dimensionless wetting potential Ω_c when the contact angle varies from 40° and 150°, the numerical result is well consistent with the analytical result so that the test cases confirm the model is reliable.



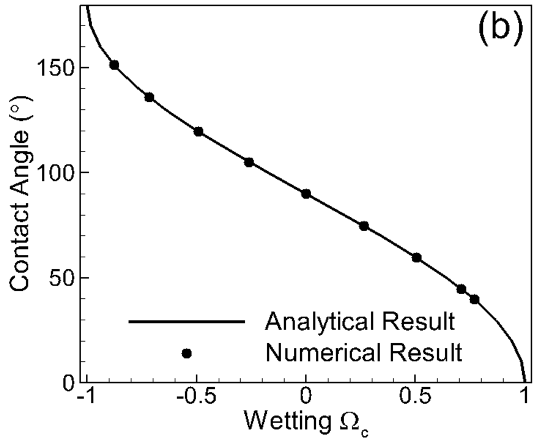


Figure 7: (a) Schematics of the half sphere bubble initially sitting on the wetting surface. The slice in the middle of domain of the equilibrium state (gray color with the red outline) is chosen to do the validation. (b) The comparison of the relationship between contact angle and dimensionless wetting potential from analytical and numerical results.

3.2. Coalescence-induced bubble detachment

In the microchannel, the gravity compared to surface tension force can be ignored so that bubbles can't easily detach from the surface. They will coalesce and eventually plug the microchannel. Bubble detachment is one of the common phenomena in bubble dynamics which happened in the post-coalescence stage needing a long time to be observed. The bubble coalescence process shows as Fig.8. It starts from the two touching parent microbubbles Fig.8(a), the neck bridge is generated and grows Fig.8(b), then bubble oscillate with small amplitude Fig.8(c), (d), (e), (f). During the oscillate process, the coalescence-induced detachment of the bubble can be observed as Fig.8(f).

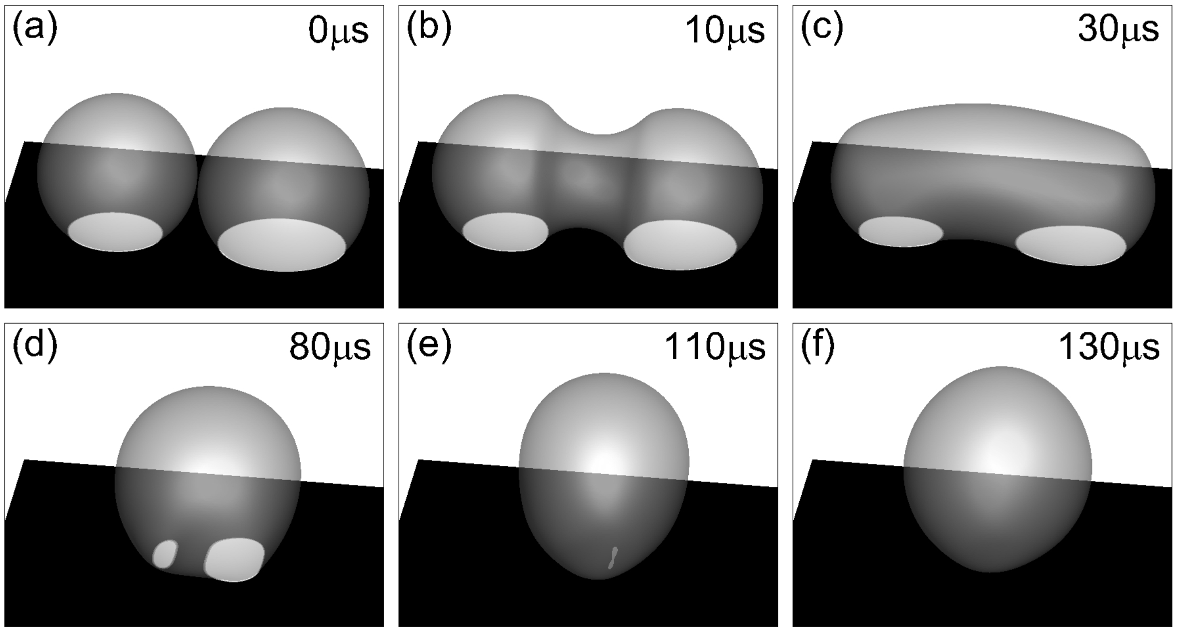


Figure 8: Time evolution of bubble coalescence

3.3. Performance test

Million lattice updates per second (MLUPS) is a parameter to indicate the computational performance for lattice Boltzmann simulations. To observe the coalescence-induced bubble detachment during two bubbles coalescing in the simulation will spend 44 days when we use the hardware: Broadwell E5-2683 v3@ 2.30GHz CPU. Additionally, massive parameter cases and multi-bubbles system are needed to study the bubble dynamics in the practical problems. Hence, GPU acceleration is critical to be employed in the application case that has high computational cost caused by a large number of cases and high resolution. The performance of our GPU accelerated lattice Boltzmann modeling is tested through single GPU and multi-GPU based on four machines with different types of CPU and GPU cards, as Tab. 2 shows. We employed the GPU parallel approaches in the same code and tested the speed-up with different machines.

	CPU	GPU	GPU number
1	Intel(R) Xeon(R) E5645	Tesla C2050	1
2	Intel(R) Xeon(R) X5660	Tesla C2750	4
3	Intel Broadwell E5-2683 v4	Tesla P100	2
4		Tesla K80	4

Table 2: Information on the machines used in the computation

- Single GPU

  GPU acceleration applied in the application case with five different spatial resolutions $88 \times 48 \times 88$, $132 \times 72 \times 132$, $176 \times 96 \times 176$, $220 \times 120 \times 220$, $264 \times 144 \times 264$. Here, we use the advancing GPU card

P100 (the third machine listed in Tab. 2), which is popularly equipped in the supercomputers to do the performance test. The speed-up increases with resolution increases. We can see from Tab. 3 the speed-up is 2110 when the appropriate resolution derived from convergence check is selected. In other words, a numerical case can be done within half an hour with parallel execution, whereas it takes 44 days with serial execution.

Resolution	GPU parallel(MLUPS)	Serial(MLUPS)	Speed-up	Parallel vs. Serial (in hour)
$88 \times 48 \times 88$	181	0.13	1428	0.02/29
$132 \times 72 \times 132$	176	0.11	1653	0.08/132
$176 \times 96 \times 176$	200	0.11	1758	0.25/439
$220 \times 120 \times 220$	211	0.10	2110	0.50/1055
$264 \times 144 \times 264$	206	0.10	1994	1.25/2492

Table 3: GPU acceleration comparing with the CPU serial performance. The last column is a comparison of parallel vs. serial wall-clock time for a complete process of coalescence.

Meanwhile, we take the appropriate resolution $220 \times 120 \times 220$ to test performance with a single GPU in different machines listed as Tab. 2. The result of the performance test is seen in Fig. 9. The advancing hardware has faster performance, such as Tesla P100 is 20 times faster than the Tesla C2050/C2750.

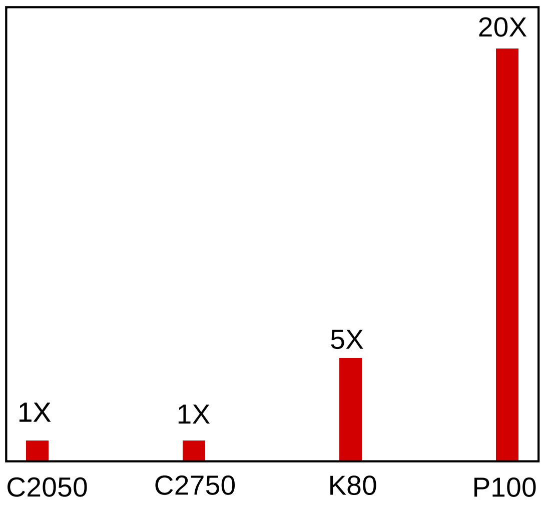


Figure 9: The performance of the different machine resources listed in the Tab.2

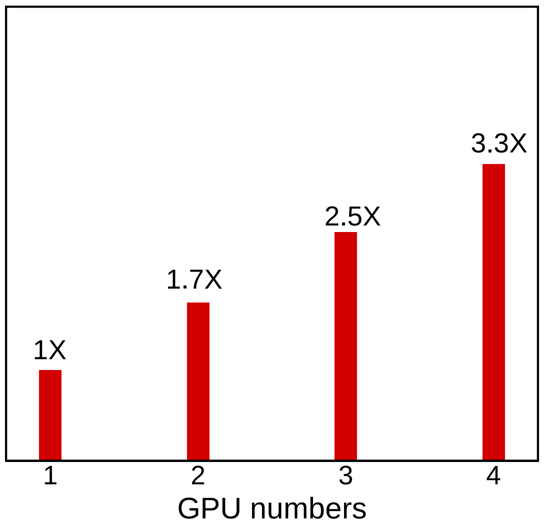


Figure 10: Performance of different numbers of GPU card. The performance accelerates as the numbers of GPU card increases.

- Multi-GPU

  Three kinds of four machines listed in Tab. 2 can be used to do the multi-GPU parallelism. Considering the number of GPU cards and the fastest performance of GPU cards, we use Tesla K80 (machine 4) to perform the efficiency of the algorithm of multi-GPU. Fig 10 exhibits the GPU acceleration with the spatial resolution 220×120×220 by one, two, three, four GPU cards separately. The performance is accelerating when the number of GPU cards increases.

4. Summary & Future Work

In this paper, we want to perform the capability and efficiency of our GPU accelerated lattice Boltzmann modeling that imposes a gas-liquid-solid interface to study the bubble dynamics. First, to perform the capability, we validate a half-sphere bubble sitting on the different wetting surfaces with a targeted contact angle from 40° to 150°. In the bubble dynamics problems, the speed accelerated is critical due to the high computational cost caused by the massive parameter cases and fine resolution. Then the efficiency of GPU accelerated lattice Boltzmann modeling is tested by the coalescence-induced bubble detachment phenomena when two parent oxygen bubbles in the hydrogen peroxide solution attached in the Pt surface, which the targeted contact angle is 36°. From the test, the performance of the lattice Boltzmann modeling is increasing up to 2110, 6963 by single GPU, 4 GPU cards accelerated, respectively. For the current application case, we use it to study the bubble dynamics, the efficiency of the modeling is enough. However, in large scale problems such as multiply bubbles system, arbitrary geometry, and so on, the efficiency of the modeling needs improving. There are many approaches we want to develop in the future: 1. Optimize the multi-GPU algorithm, especially on the part of data transformation and distribution; 2. Develop the approach that MPI, OpenMP combines with GPU parallelism with one node and multi-nodes.

Acknowledgments

This research is supported by the graduate assistantship from School of Engineering and Technology. And this work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

References

[1] W. Li, X. Wei, A. Kaufman, Implementing lattice Boltzmann computation on graphics hardware, The Visual Computer 19 (7) (2003) 444–456.

[2] NVIDIA Corporation, CUDA Zone, `https://developer.nvidia.com/cuda-zone`.

[3] S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, W. Hwu, Optimization principles and application performance evaluation of a multithreaded GPU using cuda, in: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, ACM, 2008, pp. 73–82.

[4] J. Henning, Spec cpu2006 benchmark descriptions. acm sigarch comput archit news 34 (4): 1–17 (2006).

[5] M. Wittmann, T. Zeiser, G. Hager, G. Wellein, Comparison of different propagation steps for lattice Boltzmann methods, Computers & Mathematics with Applications 65 (6) (2013) 924–935.

[6] J. Tölke, M. Krafczyk, TeraFLOP computing on a desktop PC with GPUs for 3D CFD, International Journal of Computational Fluid Dynamics 22 (7) (2008) 443–456.

[7] J. Tölke, Implementation of a lattice Boltzmann kernel using the compute unified device architecture developed by NVIDIA, Computing and Visualization in Science 13 (1) (2010) 29–39.

[8] P. Bailey, J. Myre, S. Walsh, D. Lilja, M. Saar, Accelerating lattice Boltzmann fluid flow simulations using graphics processors, in: Parallel Processing, 2009. ICPP'09. International Conference on, IEEE, 2009, pp. 550–557.

[9] C. Obrecht, F. Kuznik, B. Tourancheau, J. Roux, A new approach to the lattice Boltzmann method for graphics processing units, Computers & Mathematics with Applications 61 (12) (2011) 3628–3638.

[10] N. Tran, M. Lee, S. Hong, Performance optimization of 3D lattice Boltzmann flow solver on a GPU, Scientific Programming 2017 (2017).

[11] G. Herschlag, S. Lee, J. S. Vetter, A. Randles, GPU data access on complex geometries for D3Q19 lattice Boltzmann method, in: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), IEEE, 2018, pp. 825–834.

[12] NVIDIA Corporation, CUDA Toolkit Documentation, `http://docs.nvidia.com/cuda/index.html`, accessed: September 2015 (2015).

[13] O. Navarro-Hinojosa, S. Ruiz-Loza, M. Alencastre-Miranda, Physically based visual simulation of the lattice Boltzmann method on the GPU: a survey, The Journal of Supercomputing 74 (2018) 3441–3467.

[14] X. Li, Y. Zhang, X. Wang, W. Ge, GPU-based numerical simulation of multi-phase flow in porous media using multiple-relaxation-time lattice Boltzmann method, Chemical Engineering Science 102 (2013) 209–219.

[15] M. Januszewski, M. Kostur, Sailfish: A flexible multi-GPU implementation of the lattice Boltzmann method, Computer Physics Communications 185 (9) (2014) 2350–2368.

[16] T. Huang, C. Chang, C. Lin, Simulation of droplet dynamic with high density ratio two-phase lattice Boltzmann model on multi-GPU cluster, Computers & Fluids 173 (2018) 80–87.

[17] C. Obrecht, F. Kuznik, B. Tourancheau, J. Roux, Multi-GPU implementation of the lattice Boltzmann method, Computers & Mathematics with Applications 65 (2) (2013) 252–261.

[18] B. Zigon, L. Zhu, F. Song, Interactive 3D simulation for fluid–structure interactions using dual coupled GPUs, The Journal of Supercomputing 74 (1) (2018) 37–64.

[19] J. Myre, S. Walsh, D. Lilja, M. Saar, Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters, Concurrency and Computation: Practice and Experience 23 (4) (2011) 332–350.

[20] W. Xian, A. Takayuki, Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster, Parallel Computing 37 (9) (2011) 521–535.

[21] S. Blair, C. Albing, A. Grund, A. Jocksch, Accelerating an mpi lattice Boltzmann code using openacc, in: Proceedings of the second workshop on accelerator programming using directives, ACM, 2015, p. 3.

[22] T. Lee, C.-L. Lin, A stable discretization of the lattice Boltzmann equation for simulation of incompressible two-phase flows at high density ratio, J. Comp. Phys. 206 (10) (2005) 16 – 47.

[23] T. Lee, P. Fischer, Eliminating parasitic currents in the lattice Boltzmann equation method for nonideal gases, Physical Review E 74 (4) (2006) 046709.

[24] T. Lee, Effects of incompressibility on the elimination of parasitic currents in the lattice Boltzmann equation method for binary fluids, Computers and Mathematics with Applications 58 (2009) 987 – 994.

[25] T. Lee, L. Liu, Lattice Boltzmann simulations of micron-scale drop impact on dry surfaces, J Comp Phys 229 (2010) 8045–8063.

[26] M. o. Carpinlioğlu, M. Gündoğdu, A critical review on pulsatile pipe flow studies directing towards future research topics, Flow Meas. Instrum. 12 (3) (2001) 163–174.

[27] J. Rowlinson, B. Widom, Molecular Theory of Capillarity, Oxford Univ. Press, Oxford, UK, 1989.

[28] D. Jamet, O. Lebaigue, N. Coutris, J. Delhaye, The second gradient method for the direct numerical simulation of liquid–vapor flows with phase change, Journal of Computational Physics 169 (2) (2001) 624–651.

[29] J. Yang, D. Hubball, M. Ward, E. Rundensteiner, W. Ribarsky, Value and relation display: Interactive visual exploration of large datasets with hundreds of dimensions, IEEE Transactions on Visualization and Computer Graphics 13 (3) (2007) 494–507.

[30] L. Liu, T. Lee, Wall free energy based polynomial boundary conditions for non-ideal gas lattice Boltzmann equation, International Journal of Modern Physics C 20 (11) (2009) 1749–1768.

[31] N. Chen, H. Yu, Mechanism of axis switching in low aspect-ratio rectangular jets, Computer & Mathematics with Applications 67 (2) (2014) 437–444.

[32] Z. Wang, N. Chen, Y. Zhao, H. Yu, GPU-accelerated lattice Boltzmann method for extracting real biomechanical geometry and volumetric boundary condition, Computers & Fluids 115 (2015) 192–200.

[33] S. An, H. Yu, Z. Wang, R. Chen, B. Kapadia, J. Yao, Unified mesoscopic modeling and GPU-accelerated computational method for image-based pore-scale porous media flows, Int. J. of Heat Mass Trans. 115 (2017) 1192 – 1202.

[34] S. An, H. Yu, J. Yao, GPU-accelerated volumetric lattice Boltzmann method for porous media flow, J. Petro. Sci. Eng. 156 (2017) 546 – 552.

[35] N. Delbosc, J. Summers, A. Khan, N. Kapur, C. Noakes, Optimized implementation of the lattice Boltzmann method on a graphics processing unit towards real-time fluid simulation, Computers & Mathematics with Applications 67 (2) (2014) 462–475.

[36] J. Power, M. Hill, D. Wood, Supporting x86-64 address translation for 100s of GPU lanes, in: 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), IEEE, 2014, pp. 568–578.